Refine Search

Search Results -

Terms	Documents
L12 and query$ and match$ near text near string near pattern$	2

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Tuesday, September 07, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L13	L12 and query$ and match$ near text near string near pattern$	2	L13
L12	(database or data with base)	483979	L12
L11	L10 and update	23	L11
L10	L9 and lists	28	L10
L9	L8 and match$	30	L9
L8	L7 and text near pattern	30	L8
L7	L6 and (text with string or text near string)	760	L7
L6	(data with base or database) near query$	8333	L6
L5	L4 and stor$ near (text with string or text near string or text near pattern)	12	L5
L4	L3 and (database or data with base)	1456	L4
L3	L2 and server	1539	L3
L2	L1 and search$	2162	L2

L1	financial near transactions	7763	L1

END OF SEARCH HISTORY

L11: Entry 9 of 23 File: USPT Sep 3, 2002

DOCUMENT-IDENTIFIER: US 6446076 B1
TITLE: Voice interactive web-based agent system responsive to a user location for prioritizing and formatting information

Drawing Description Text (6):
FIG. 4 is a flowchart for pattern matching in accordance with a preferred embodiment;

Detailed Description Text (23):
Polymorphism and multiple inheritance make it possible for different programmers to mix and match characteristics of many different classes and create specialized objects that can still work with related objects in predictable ways.

Detailed Description Text (46):
In accordance with a preferred embodiment, BackgroundFinder (BF) is implemented as an agent responsible for preparing an individual for an upcoming meeting by helping him/her retrieve relevant information about the meeting from various sources. BF receives input text in character form indicative of the target meeting. The input text is generated in accordance with a preferred embodiment by a calendar program that includes the time of the meeting. As the time of the meeting approaches, the calendar program is queried to obtain the text of the target event and that information is utilized as input to the agent. Then, the agent parses the input meeting text to extract its various components such as title, body, participants, location, time etc. The system also performs pattern matching to identify particular meeting fields in a meeting text. This information is utilized to query various sources of information on the web and obtain relevant stories about the current meeting to send back to the calendaring system. For example, if an individual has a meeting with Netscape and Microsoft to talk about their disputes, and would obtain this initial information from the calendaring system. It will then parse out the text to realize that the companies in the meeting are "Netscape" and "Microsoft" and the topic is "disputes." Then, the system queries the web for relevant information concerning the topic. Thus, in accordance with an objective of the invention, the system updates the calendaring system and eventually the user with the best information it can gather to prepare the user for the target meeting. In accordance with a preferred embodiment, the information is stored in a file that is obtained via selection from a link imbedded in the calendar system.

Detailed Description Text (48):
A computer program in accordance with a preferred embodiment is organized in five distinct modules: BF.Main, BF.Parse, Background Finder.Error, BF.PatternMatching and BF.Search. There is also a frmMain which provides a user interface used only for debugging purposes. The executable programs in accordance with a preferred embodiment never execute with the user interface and should only return to the calendaring system through Microsoft's Winsock control. A preferred embodiment of the system executes in two different modes which can be specified under the command line sent to it by the calendaring system. When the system runs in simple mode, it executes a keyword query to submit to external search engines. When executed in complex mode, the system performs pattern matching before it forms a query to be sent to a search engine.

Detailed Description Text (51):
The user-defined structure, tMeetingRecord, is used to store all the pertinent information concerning a single meeting. This info includes userID, an original description of the meeting, the extracted list of keywords from the title and body of meeting etc. It is important to note that only one meeting record is created per instance of the system in accordance with a preferred embodiment. This is because each time the system is spawned to service an upcoming meeting, it is assigned a task to retrieve information for only one meeting. Therefore, the meeting record created corresponds to the current meeting examined. ParseMeetingText populates this meeting record and it is then passed around to provide information about the meeting to other functions. If GoPatternMatch can bind any values to a particular meeting field, the corresponding entries in the meeting record is also updated. The structure of tMeetingRecord with each field described in parentheses is provided below in accordance with a preferred embodiment.

Detailed Description Text (52):
There are two other structures which are created to hold each individual pattern utilized in pattern matching. The record tAPatternRecord is an array containing all the components/elements of a pattern. The type tAPatternElement is an array of strings which represent an element in a pattern. Because there may be many "substitutes" for each element, we need an array of strings to keep track of what all the substitutes are. The structures of tAPatternElement and tAPatternRecord are presented below in accordance with a preferred embodiment. Public Type tAPatternElement elementArray( ) As String End Type Public Type tAPatternRecord patternArray( ) As tAPatternElement End Type

Detailed Description Text (54):
Many constants are defined in each declaration section of the program which may need to be updated periodically as part of the process of maintaining the system in accordance with a preferred embodiment. The constants are accessible to allow dynamic configuration of the system to occur as updates for maintaining the code.

Detailed Description Text (55):
Included in the following tables are lists of constants from each module which I thought are most likely to be modified from time to time. However, there are also other constants used in the code not included in the following list. It does not mean that these non-included constants will never be changed. It means that they will change much less frequently.

Detailed Description Text (59):
For Pattern Matching Module (BFPatternMatch): There are no constants in this module which require frequent updates.

Detailed Description Text (61):
The best way to depict the process flow and the coordination of functions between each other is with the five flowcharts illustrated in FIGS. 2 to 6. FIG. 2 depicts the overall process flow in accordance with a preferred embodiment. Processing commences at the top of the chart at function block 200 which launches when the program starts. Once the application is started, the command line is parsed to remove the appropriate meeting text to initiate the target of the background find operation in accordance with a preferred embodiment as shown in function block 210. A global stop list is generated after the target is determined as shown in function block 220. Then, all the patterns that are utilized for matching operations are generated as illustrated in function block 230. Then, by tracing through the chart, function block 200 invokes GoBF 240 which is responsible for logical processing associated with wrapping the correct search query information for the particular target search engine. For example, function block 240 flows to function block 250 and it then calls GoPatternMatch as shown in function block 260. To see the process flow of GoPatternMatch, we swap to the diagram titled "Process Flow for BF's Pattern Matching Unit."

Detailed Description Text (62):
One key thing to notice is that functions depicted at the same level of the chart are called by in sequential order from left to right (or top to bottom) by their common parent function. For example, Main 200 calls ProcessCommandLine 210, then CreateStopListist 220, then CreatePatterns 230, then GoBackgroundFinder 240. FIGS. 3 to 6 detail the logic for the entire program, the parsing unit, the pattern matching unit and the search unit respectively. FIG. 6 details the logic determinative of data flow of key information through BackgroundFinder, and shows the functions that are responsible for creating or processing such information.

Detailed Description Text (70):
FIG. 3 is a user profile data model in accordance with a preferred embodiment. Processing commences at function block 300 which is responsible for invoking the program from the main module. Then, at function block 310, a wrapper function is invoked to prepare for the keyword extraction processing in function block 320. After the keywords are extracted, then processing flows to function block 330 to determine if the delimiters are properly positioned. Then, at function block 340, the number of words in a particular string is calculated and the delimiters for the particular field are and a particular field from the meeting text is retrieved at function block 350. Then, at function block 380, the delimiters of the string are again checked to assure they are placed appropriately. Finally, at function block 360, the extraction of each word from the title and body of the message is performed a word at a time utilizing the logic in function block 362 which finds the next closest word delimiter in the input phrase, function block 364 which strips unnecessary materials from a word and function block 366 which determines if a word is on the stop list and returns an error if the word is on the stop list.

Detailed Description Text (71):
Pattern Matching in Accordance with a Preferred Embodiment

Detailed Description Text (77):
Pattern Matching Overcomes these Limitations in Accordance with a Preferred Embodiment

Detailed Description Text (78):
Here's how the pattern matching system can address each of the corresponding issues above in accordance with a preferred embodiment.

Detailed Description Text (79):
1. By doing pattern matching, we match up only parts of the meeting text that we want and extract those parts.

Detailed Description Text (80):
2. By performing pattern matching on the meeting body and extracting only the parts from the meeting body that we want. Our meeting body will not go to complete waste then.

Detailed Description Text (81):
3. Pattern matching is based on a set of templates that we specify, allowing us to identify people names, company names etc from a meeting text.

Detailed Description Text (82):
4. In summary, with pattern matching, we no longer suffer from information overload. Of course, the big problem is how well our pattern matching works. If we rely exclusively on artificial intelligence processing, we do not have a 100% hit rate. We are able to identify about 20% of all company names presented to us.

Detailed Description Text (85):
Pattern Matching Terminology

Detailed Description Text (86):
The common terminology associated with pattern matching is provided below.

Detailed Description Text (95):
Here is a table representing all the patterns supported by BF. Each pattern belongs to a pattern group. All patterns within a pattern group share a similar format and they only differ from each other in terms of what indicators are used as substitutes. Note that the patterns which are grayed out are also commented in the code. BF has the capability to support these patterns but we decided that matching these patterns is not essential at this point.

Detailed Description Text (96):
FIG. 4 is a detailed flowchart of pattern matching in accordance with a preferred embodiment. Processing commences at function block 400 where the main program invokes the pattern matching application and passes control to function block 410 to commence the pattern match processing. Then, at function block 420, the wrapper function loops through to process each pattern which includes determining if a part of the text string can be bound to a pattern as shown in function block 430. Then, at function block 440, various placeholders are bound to values if they exist, and in function block 441, a list of names separated by punctuation are bound, and at function block 442 a full name is processed by finding two capitalized words as a full name and grabbing the next letter after a space after a word to determine if it is capitalized. Then, at function block 443, time is parsed out of the string in an appropriate manner and the next word after a blank space in function block 444. Then, at function block 445, the continuous phrases of capitalized words such as company, topic or location are bound and in function block 446, the next word after the blank is obtained for further processing in accordance with a preferred embodiment. Following the match meeting field processing, function block 450 is utilized to loacte an indicator which is the head of a pattern, the next word after the blank is obtained as shown in function block 452 and the word is checked to determine if the word is an indicator as shown in function block 454. Then, at function block 460, the string is parsed to locate an indicator which is not at the end of the pattern and the next word after unnecessary white space such as that following a line feed or a carriage return is processed as shown in function block 462 and the word is analyzed to determine if it is an indicator as shown in function block 464. Then, in function block 470, the temporary record is reset to the null set to prepare it for processing the next string and at function block 480, the meeting record is updated and at function block 482 a check is performed to determine if an entry is already made to the meeting record before parsing the meeting record again.

Detailed Description Text (98):
Now that we have identified fields within the meeting text which we consider important, there are quite a few things we can do with it. One of the most important applications of pattern matching is of course to improve the query we construct which eventually gets submitted to Alta Vista and News Page. There are also a lot of other options and enhancements which exploit the results of pattern matching that we can add to BF. These other options will be described in the next section. The goal of this section is to give the reader a good sense of how the results obtained from pattern matching can be used to help us obtain better search results.

Detailed Description Text (105):
FIG. 6 is a flowchart of the actual code utilized to prepare and submit searches to the Alta Vista and Newspage search engines in accordance with a preferred embodiment. Processing commences at function block 610 where a command line is utilized to update a calendar entry with specific calendar information. The message is next posted in accordance with function block 620 and a meeting record is created to store the current meeting information in accordance with function block

630. Then, in function block 640 the query is submitted to the Alta Vista search engine and in function block 650, the query is submitted to the Newspage search engine. When a message is returned from the search engine, it is stored in a results data structure as shown in function block 660 and the information is processed and stored in summary form in a file for use in preparation for the meeting as detailed in function block 670.

Detailed Description Text (109):
Enhance Target Rate for Pattern Matching

Detailed Description Text (111):
Depending on what location the system identifies through pattern matching or alternatively depending on what location the user indicates as the meeting place, a system in accordance with a preferred embodiment suggests a plurality of fine restaurants whenever it detects the words lunch/dinner/breakfast. We can also use a site like company finder to confirm what we got is indeed a company name or if there is no company name that pattern matching can identify, we can use a company finder web site as a "dictionary" for us to determine whether certain capitalized words represent a company name. We can even display stock prices and breaking news for a company that we have identified.

Detailed Description Text (118):
mySite! is a high-impact, Internet-based application in accordance with a preferred embodiment that is focused on the theme of delivering services and providing a personalized experience for each customer via a personal web site in a buyer-centric world. The services are intuitively organized around satisfying customer intentions-fundamental life needs or objectives that require extensive planning decisions, and coordination across several dimensions, such as financial planning, healthcare, personal and professional development, family life, and other concerns. Each member owns and maintains his own profile, enabling him to create and browse content in the system targeted specifically at him. From the time a demand for products or services is entered, to the completion of payment, intelligent agents are utilized to conduct research, execute transactions and provide advice. By using advanced profiling and filtering, the intelligent agents learn about the user, improving the services they deliver. Customer intentions include Managing Daily Logistics (e.g., email, calendar, contacts, to-do list, bill payment, shopping, and travel planning); and Moving to a New Community (e.g., finding a place to live, moving household possessions, getting travel and shipping insurance coverage, notifying business and personal contacts, learning about the new community). From a consumer standpoint, mySite! provides a central location where a user can access relevant products and services and accomplish daily tasks with ultimate ease and convenience.

Detailed Description Text (121):
An Egocentric Interface is a user interface crafted to satisfy a particular user's needs, preferences and current context. It utilizes the user's personal information that is stored in a central profile database to customize the interface. The user can set security permissions on and preferences for interface elements and content. The content integrated into the Egocentric Interface is customized with related information about the user. When displaying content, the Egocentric Interface will include the relationship between that content and the user in a way that demonstrates how the content relates to the user. For instance, when displaying information about an upcoming ski trip the user has signed up for, the interface will include information about events from the user's personal calendar and contact list, such as other people who will be in the area during the ski trip. This serves to put the new piece of information into a context familiar to the individual user.

Detailed Description Text (124):
The Supplier Profile Database 1050 contains information about the product and

service providers integrated into the intention. The information contained in this database provides a link between the intention framework and the suppliers. It includes product lists, features and descriptions, and addresses of the suppliers' product web sites. The Customer Profile Database 1060 contains personal information about the customers, such as name, address, social security number and credit card information, personal preferences, behavioral information, history, and web site layout preferences. The Supplier's Web Server 1070 provides access to all of the supplier's databases necessary to provide information and transactional support to the customer.

Detailed Description Text (126):
FIG. 10B is a flowchart providing the logic utilized to create a web page within the Egocentric Interface. The environment assumes a web server and a web browser connected through a TCP/IP network, such as over the public Internet or a private Intranet. Possible web servers could include Microsoft Internet Information Server, Netscape Enterprise Server or Apache. Possible web browsers include Microsoft Internet Explorer or Netscape Navigator. The client (i.e. web browser) makes a request 1001 to the server (i.e. web server) for a particular web page. This is usually accomplished by a user clicking on a button or a link within a web page. The web server gets the layout and content preferences 1002 for that particular user, with the request to the database keyed off of a unique user id stored in the client (i.e. web browser) and the User profile database 1003. The web server then retrieves the content 1004 for the page that has been requested from the content database 1005. The relevant user-centric content, such as calendar, email, contact list, and task list items are then retrieved 1006. (See FIG. 11 for a more detailed description of this process.) The query to the database utilizes the user content preferences stored as part of the user profile in the User profile database 1003 to filter the content that is returned. The content that is returned is then formatted into a web page 1007 according to the layout preferences defined in the user profile. The web page is then returned to the client and displayed to the user 1008.

Detailed Description Text (127):
FIG. 11 describes the process of retrieving user-centric content to add to a web page. This process describes 1006 in FIG. 10B in a more detailed fashion. It assumes that the server already has obtained the user profile and the existing content that is going to be integrated into this page. The server parses 1110 the filtered content, looking for instances of events, contact names and email addresses. If any of these are found, they are tagged and stored in a temporary holding space. Then, the server tries to find any user-centric content 1120 stored in various databases. This involves matching the tagged items in the temporary storage space with calendar items 1130 in the Calendar Database 1140; email items 1115 in the Email Database 1114; contact items 1117 in the Contact Database 1168; task list items 1119 in the Task List Database 1118; and news items 1121 in the News Database 1120. After retrieving any relevant user-centric content, it is compiled together and returned 1122.

Detailed Description Text (131):
FIG. 13 describes the data model that supports the Persona concept. The user table 1310 contains a record for each user who has an account in the system. This table contains a username and a password 1320 as well as a unique identifier. Each user can have multiple Personas 1330, which act as containers for more specialized structures called Profiles 1340. Profiles contain the detailed personal information in Profile Field 1350 records. Attached to each Profile are sets of Profile Restriction 1360 records. These each contain a Name 1370 and a Rule 1380, which define the restriction. The Rule is in the form of a pattern like (if x then y), which allows the Rule to be restricted to certain uses. An example Profile Restriction would be the rule that dictates that the user cannot book a flight on a certain airline contained in the list. This Profile Restriction could be contained in the "Travel" Profile of the "Work" Persona set up by the user's employer, for

instance. Each Profile Field also contains a set of Permissions 1390 that are contained in that record. These permissions dictate who has what access rights to that particular Profile Field's information.

Detailed Description Text (140):
The system provide Consumer Report-like service that is customized for each user based on a user profile. The system records and provides ratings from users about product quality and desirability on a number of dimensions. The difference between this system and traditional product quality measurement services is that the ratings that come back to the users are personalized. This service works by finding the people who have the closest match to the user's profile and have previously rated the product being asked for. Using this algorithm will help to ensure that the product reports sent back to the user only contain statistics from people who are similar to that user.

Detailed Description Text (141):
FIG. 16 describes the algorithm for determining the personalized product ratings for a user. When the user requests a product report 1610 for product X, the algorithm retrieves the profiles 1620 from the profile database 1630 (which includes product ratings) of those users who have previously rated that product. Then the system retrieves the default thresholds 1640 for the profile matching algorithm from the content database 1650. It then maps all of the short list of users along several dimensions specified in the profile matching algorithm 1660. The top n (specified previously as a threshold variable) nearest neighbors are then determined and a test is performed to decide if they are within distance y (also specified previously as a threshold variable) of the user's profile in the set 1670 using the results from the profile matching algorithm. If they are not within the threshold, then the threshold variables are relaxed 1680, and the test is run again. This processing is repeated until the test returns true. The product ratings from the smaller set of n nearest neighbors are then used to determine a number of product statistics 1690 along several dimensions. Those statistics are inserted into a product report template 1695 and returned to the user 1697 as a product report.

Detailed Description Text (143):
This system provides one central storage place for a person's profile. This storage place is a server available through the public Internet, accessible by any device that is connected to the Internet and has appropriate access. Because of the ubiquitous accessibility of the profile, numerous access devices can be used to customize services for the user based on his profile. For example, a merchant's web site can use this profile to provide personalized content to the user. A Personal Digital Assistant (PDA) with Internet access can synchronize the person's calendar, email, contact list, task list and notes on the PDA with the version stored in the Internet site. This enables the person to only have to maintain one version of this data in order to have it available whenever it is needed and in whatever formats it is needed.

Detailed Description Text (148):
FIG. 18 discloses the detailed interaction between a consumer and the integrator involving one supplier. The user accesses a Web Browser 1810 and requests product and pricing information from the integrator. The request is sent from the user's browser to the integrator's Web/Application Server 1820. The user's preferences and personal information is obtained from an integrator's customer profile database 1830 and returned to the Web/Application server. The requested product information is extracted from the supplier's product database 1840 and customized for the particular customer. The Web/Application server updates the supplier's customer information database 1850 with the inquiry information about the customer. The product and pricing information is then formatted into a Web Page 1860 and returned to the customer's Web Browser.

Detailed Description Text (151):
FIG. 19 discloses the logic in accordance with a preferred embodiment processing by an agent to generate a verbal summary for the user. When the user requests the summary page 1900, the server gets the user's agent preferences 1920, such as agent type, rules and summary level from the user profile database 1930. The server gets the content 1940, such as emails, to do list items, news, and bills, from the content database 1950. The agent parses all of this content, using the rules stored in the profile database, and summarizes the content 1960. The content is formatted into a web page 1970 according to a template. The text for the agent's speech is generated 1980, using the content from the content database 1990 and speech templates stored in the database. This speech text is inserted into the web page 1995 and the page is returned to the user 1997.

Detailed Description Text (160):
An Event Backgrounder is a short description of an upcoming event that is sent to the user just before an event. The Event Backgrounder is constantly updated with the latest information related to this event. Pertinent information such as itinerary and logistics are included, and other useful information, such as people the user knows who might be in the same location, are also included. The purpose of the Event Backgrounder is to provide the most up-to-date information about an event, drawing from a number of resources, such as public web sites and the user's calendar and contact lists, to allow the user to react optimally in a given situation.

Detailed Description Text (162):
This software looks for opportunities to tell the user when a friend, family member or acquaintance is or is going to be in the same vicinity as the user. This software scans the user's calendar for upcoming events. It then uses a geographic map to compare those calendar events with the calendar events of people who are listed in his contact list. It then informs the user of any matches, thus telling the user that someone is scheduled to be near him at a particular time.

Detailed Description Text (173):
The Awareness Machine is a combination of hardware device and software application. The hardware consists of handheld personal computer and wireless communications device. The Awareness Machine reflects a constantly updated state-of-the-owner's-world by continually receiving a wireless trickle of information. This information, mined and processed by a suite of intelligent agents, consists of mail messages, news that meets each user's preferences, schedule updates, background information on upcoming meetings and events, as well as weather and traffic.

Detailed Description Paragraph Table (1):
A.1.1.1.1.1 Public Type tMeetingRecord sUserID As String (user id given by Munin) sTitleOrig As String (original non stop listed title we need to keep around to send back to Munin) sTitleKW As String (stoplisted title with only keywords) sBodyKW As String (stoplisted body with only keywords) sCompany() As String (companys identified in title or body through pattern matching) sTopic() As String (topics identified in title or body through pattern matching) sPeople() As String (people identified in title or body through pattern matching) sWhen() As String (time identified in title or body through pattern matching) sWhere() As String (location identified in title or body through pattern matching) sLocation As String (location as passed in by Munin) sTime As String (time as passed in by Munin) sParticipants() As String (all participants engaged as passed in by Munin) sMeetingText As String (the original meeting text w/o userid) End Type

Detailed Description Paragraph Table (4):
CURRENT CONSTANT VALUE USE PORTION.sub.-- "::" Define the separator between SEPARATOR different portions of the meeting text sent in by Munin. For example in "09::Meet with Chad::about life::Chad .vertline. Denise::::::" "::" is the separator between different parts of the meeting text PARTICIPANT.sub.--

".vertline." Define the separator between each SEPARATOR participant in the participant list portion of the original meeting text Refer to example above.

Detailed Description Paragraph Table (7):
Procedure Name Type Called By Description Main Public None This is the main function (BF.Main) Sub where the program first launches. It initializes BF with the appropriate parameters (e.g. Internet time- out, stoplist...) and calls GoBF to launch the main part of the program. ProcessCom Private Main This function parses the mandLine Sub command line. It assumes (BF.Main) that the delimiter indicating the beginning of input from Munin is stored in the constant CMD_SEPARATOR. CreateStopLi Private Main This function sets up a stop st Func- list for future use to parse out (BF.Main) tion unwanted words from the meeting text. There are commas on each side of each word to enable straight checking. Create Public Main This procedure is called once Patterns Sub when BF is first initialized to (BF.Pattern create all the potential Match) patterns that portions of the meeting text can bind to. A pattern can contain however many elements as needed. There are two types of elements. The first type of elements are indicators. These are real words which delimit the potential of a meeting field (eg company) to follow. Most of these indicators are stop words as expected because stop words are words usually common to all meeting text so it makes sense they form patterns. The second type of elements are special strings which represent placeholders. A placeholder is always in the form of $*$ where * can be either PEOPLE, COMPANY, TOPIC_UPPER, TIME, LOCATION or TOPIC_ALL. A pattern can begin with either one of the two types of elements and can be however long, involving however any number/type of elements. This procedure dynamically creates a new pattern record for each pattern in the table and it also dynamically creates new tAPatternElements for each element within a pattern. In addition, there is the concept of being able to substitute indicators within a pattern. For example, the pattern $PEOPLE$ of $COMPANY$ is similar to the pattern $PEOPLE$ from $COMPANY$. "from" is a substitute for "of". Our structure should be able to express such a need for substitution. GoBF Public Main This is a wrapper procedurer (BF.Main) Sub that calls both the parsing and the searching subroutines of the BF. It is also responsible for sending data back to Munin. ParseMeetin Public GoBackGroundF This function takes the initial gText Func- inder meeting text and identifies (BF.Parse) tion the userID of the record as well as other parts of the meeting text including the title, body, participant list, location and time. In addition, we call a helper function ProcessStopList to eliminate all the unwanted words from the original meeting title and meeting body so that only keywords are left. The information parsed out is stored in the MeetingRecord structure. Note that this function does no error checking and for the most time assumes that the meeting text string is correctly formatted by Munin. The important variable is thisMeeting Record is the temp holder for all info regarding current meeting. It's eventually returned to caller. FormatDelim Private ParseMeetingTe There are 4 ways in which itation xt, the delimiters can be placed. (BF.Parse) DetermineNum We take care of all these Words, cases by reducing them GetAWordFrom down to Case 4 in which String there are no delimiters around but only between fields in a string (e.g. A::B::C) Determine Public ParseMeeting This functions determines NumWords Func- Text, how many words there are in (BF.Parse) tion ProcessStop a string (stInEvalString) The List function assumes that each word is separated by a designated separator as specified in stSeparator. The return type is an integer that indicates how many words have been found assuming each word in the string is separated by stSeparator. This function is always used along with GetAwordFromString and should be called before calling GetAWordFrom String. GetAWordFr Public ParseMeeting This function extracts the ith omString Func- Text, word of the (BF.Parse) tion ProcessStop string(stInEvalstring) List assuming that each word in the string is separated by a designated separator contained in the variable stSeparator. In most cases, use this function with DetermineNumWords. The function returns the wanted word. This function checks to make sure that iInWordNum is within bounds so that is not greater than the total number ofwords in string or less than/equal to zero. If it is out of bounds, we return empty string to indicate we can't get anything. We try to make sure this doesn't happen by calling

DetermineNumWords first. ParseAndCle Private ParseMeetingTe This function first grabs the anPhrase Func- xt word and send it to (BF .Parse) tion CleanWord in order strip the stuff that nobody wants. There are things in parseWord that will kill the word, so we will need a method of looping through the body and rejecting words without killing the whole function I guess keep CleanWord and check a return value ok, now I have a word so I need to send it down the parse chain. This chain goes ParseCleanPhrase -> CleanWord -> EvaluateWord. If the word gets through the entire chain without being killed, it will be added at the end to our keyword string. first would be the function that checks for "/" as a delimiter and extracts the parts of that. This I will call "StitchFace" (Denise is more normal and calls it GetAWordFromString) if this finds words,then each of these will be sent, in turn, down the chain. If these get through the entire chain without being added or killed then they will be added rather than tossed. FindMin Private ParseAndCleanP This function takes in 6 input (BF.Parse) Func- hrase values and evaluates to see tion what the minimum non zero value is. It first creates an array as a holder so that we can sort the five input values in ascending order. Thus the minimum value will be the first non zero value element of the array. If we go through entire array without finding a non zero value, we know that there is an error and we exit the function. CleanWord Private ParseAndCleanP This function tries to clean (BF.Parse) Func- hrase up a word in a meeting text. tion It first of all determines if the string is of a valid length. It then passes it through a series oftests to see it is clean and when needed, it will edit the word and strip unnecessary characters off of it. Such tests includes getting rid of file extensions, non chars, numbers etc. EvaluateWor Private ParseAndCleanP This function tests to see if d Func- hrase this word is in the stop list so (BF.Parse) tion it can determine whether to eliminate the word from the original meeting text. If a word is not in the stoplist, it should stay around as a keyword and this function exits beautifully with no errors. However, if the words is a stopword, an error must be returned. We must properly delimit the input test string so we don't accidentally retrieve sub strings. GoPatternMa Public GoBF This procedure is called tch Sub when our QueryMethod is (BF.Pattern set to complex query Match) meaning we do want to do all the pattern matching stuff. It

Detailed Description Paragraph Table (8):
's a simple wrapper function which initializes some arrays and then invokes pattern matching on the title and the body. MatchPattern Public GoPattern Match This procedure loops through s Sub every pattern in the pattern (BF.Pattern table and tries to identify Match) different fields within a meeting text specified by sInEvalString. For debugging purposes it also tries to tabulate how many times a certain pattern was triggered and stores it in gTabulateMatches to see whichp pattern fired the most. gTabulateMatches is stored as a global because we want to be able to run a batch file of 40 or 50 test strings and still be able to know how often a pattern was triggered. MatchAPatte Private MatchPatterns This function goes through m Func- each element in the current (BF.Pattern tion pattern. It first evaluates to Match) determine whether element is a placeholder or an indicator. If it is a placeholder, then it will try to bind the placeholder with some value. If it is an indicator, then we try to locate it. There is a trick however. Depending on whether we are at current element is the head of the pattern or not we want to take different actions. If we are at the head, we want to look for the indicator or the placeholder. If we can't find it, then we know that the current pattern doesn't exist and we quit. However, if it is not the head, then we continue looking, because there may still be a head somewhere. We retry in this case. etingField Private MatchAPattern This function uses a big (BF.Pattern Func- switch statement to first Match) tion determine what kind of placeholder we are talking about and depending on what type of placeholder, we have specific requirements and different binding criteria as specified in the subsequent functions called such as BindNames, BindTime etc. If binding is successful we add it to our guessing record. BindNames Private MatchMeetingFi In this function, we try to (BF.Pattern Func- eld match names to the Match) tion corresponding placeholder $PEOPLE$. Names are defined as any consecutive two words which are capitalized. We also what to retrieve a series

of names which are connected by and, or & so we look until we don't see any of these 3 separators anymore. Note that we don't want to bind single word names because it is probably too general anyway so we don't want to produce broad but irrelevant results. This function calls BindAFullName which binds one name so in a since BindNames collects all the results from BindAFullName BindAFull Private BindNames This function tries to bind a Name Func- full name. If the $PEOPLE$ (BF.Pattern tion placeholder is not the head of Match) the pattern, we know that it has to come right at the beginning of the test string because we've been deleting stuff off the head of the string all along. If it is the head, we search until we find something that looks like a full name. If we can't find it, then there's no such pattern in the text entirely and we quit entirely from this pattern. This should eventually return us to the next pattern in MatchPatterns. GetNextWor Private BindAFull This function grabs the next dAfterWhite Func- Name, word in a test string. It looks Space tion BindTime, for the next word after white (BF.Pattern BindCompanyTo spaces, @ or/. The word is Match) picLoc defined to end when we encounter another one of these white spaces or separators. BindTime Private MatchMeetingFi Get the immediate next word (BF.Pattern Func- eld and see if it looks like a time Match) tion pattern. If so we've found a time and so we want to add it to the record. We probably should add more time patterns. But people don't seem to like to enter the time in their titles these days especially since we now have tools like OutLook. BindCompan Private MatchMeetingFi This function finds a yTopicLoc Func- eld continuous capitalized string (BF.Pattern tion and binds it to stMatch Match) which is passed by reference from MatchMeetingField. A continous capitalized string is a sequence of capitalized words which are not interrupted by things like, . etc. There's probably more stuff we can add to the list of interruptions. LocatePatter Private MatchAPattern This function tries to locate nHead Func- an element which is an (BF.Pattern tion indicator. Note that this Match) indicator SHOULD BE AT THE HEAD of the pattern otherwise it would have gone to the function LocateIndicator instead. Therefore, we keep on grabbing the next word until either there's no word for us to grab (quit) or if we find one of the indicators we are looking for. ContainInArr Private LocatePattern `This function is really ay Func- Head, simple. It loops through all (BF.Pattern tion LocateIndicator the elements in the array Match) `to find a matching string. Locate Private MatchAPattern This function tries to locate Indicator Func- an element which is an (BF.Pattern tion indicator. Note that this Match) indicator is NOT at the head of the pattern otherwise it would have gone to LocatePatternHead instead. Because of this, if our pattern is to be satisfied, the next word we grab HAS to be the indicator or else we would have failed. Thus we only grab one word, test to see if it is a valid indicator and then return result. InitializeGue Private MatchAPattern This function reinitializes our ssesRecord Sub temporary test structure (BF.Pattern because we have already Match) transfered the info to the permanent structure, we can reinitialize it so they each have one element AddToMeeti Private MatchAPattern This function is only called ngRecord Sub when we know that the (BF.Pattern information stored in Match) tInCurrGuesses is valid meaning that it represents legitamate guesses of meeting fields ready to be stored in the permanent record,tInMeetingRecord. We check to make sure that we do not store duplicates and we also what to clean up what we want to store so that there's no cluttered crap such as punctuations, etc. The reason why we don't clean up until now is to save time. We don't waste resources calling ParseAndCleanPhrase until we know for sure that we are going to add it permanently. NoDuplicate Private AddToMeetingR This function loops through Entry Func- ecord each element in the array to (BF.Pattern tion make sure that the test string Match) aString is not the same as any of the strings already stored in the array. Slightly different from ContainInArray. SearchAlta Public GoBackGroundF This function prepares a Vista Func- inder query to be submited to (BF.Search) tion AltaVista Search engine. It submits it and then parses the returning result in the appropriate format containing the title, URL and body/summary of each story retrieved. The number of stories retrieved is specified by the constant NUM_AV_STORIES. Important variables include stURLAltaVista used to store query to submit stResultHTML used to store html from page specified by stURLAltaVista. ConstructAlt Private SearchAltaVista This function constructs the

aVistaURL Func- URL string for the alta vista (BF.Search) tion search engine using the advanced query search mode. It includes the keywords to be used, the language and how we want to rank the search. Depending on whether we want to use the results of our pattern matching unit, we construct our query differently. ConstructSi Private ConstructAltaVi This function marches down mpleKey Func- staURL, the list of keywords stored in Word tion ConstructNewsP the stTitleKW or stBodyKW (BF.Search) ageURL fields of the input meeting record and links them up into one string with each keyword separated by a connector as determined by the input variable stInConnector. Returns this newly constructed string. ConstructCo Private ConstructAltaVi This function constructs the

Detailed Description Paragraph Table (9):
mpleXA Func- staURL keywords to be send to the VKeyWord tion AltaVista site. Unlike (BF.Search) ConstructSimpleKeyWord which simply takes all the keywords from the title to form the query, this function will look at the results of BF 's pattern matching process and see if we are able to identify any specific company names or topics for constructing the queries. Query will include company and topic identified and default to simple query if we cannot identify either company or topic. JoinWithCon Private ConstructCompl This function simply replaces nectors Func- exA VKey the spacesbetween the words (BF.Search) tion Word, within the string with a ConstructCompl connector which is specified exNPKey by the input. Word, RefineWith Rank RefineWithD Private ConstructAltaVi This function constructs the ate (NOT Func- staURL date portion of the alta vista CALLED tion query and returns this portion AT THE of the URL as a string. It MOMENT) makes sure that alta vista (BF.Search) searches for articles within the past PAST_NDAYS. RefineWithR Private ConstructAltaVi This function constructs the ank Func- staURL string needed to passed to (BF.Search) tion Altavista in order to rank an advanced query search. If we are constructing the simple query we will take in all the keywords from the title. For the complex query, we will take in words from company and topic, much the same way we formed the query in ConstructComplexA VKeyW ord. IdentifyBloc Public SearchAltaVista, This function extracts the k Func- SearchNewsPage block within a string marked (BF.Parse) tion by the beginning and the ending tag given as inputs starting at a certain location(iStart). The block retrieved does not include the tags themselves. If the block cannot be identified with the specified delimiters, we return unsuccessful through the parameter iReturnSuccess passed to use by reference. The return type is the block retrieved. IsOpenURL Public SearchAltaVista, This function determines Error Func- SearchNewsPage whether the error (BF.Error) tion encountered is that of a timeout error. It restores the mouse to default arrow and then returns true if it is a time out or false otherwise. SearchNews Public GoBackGroundF This function prepares a Page Func- inder query to be submited to (BF. Search) tion NewsPage Search engine. It submits it and then parses the returning result in the appropriate format containing the title, URL and body/summary of each story retrieved. The number of stories retrieved is specified by the constant UM_NP_STORIES ConstructNe Private SearchNewsPage This function constructs the wsPageURL Func- URL to send to the (BF.Search) tion NewsPage site. It uses the information contained in the input meeting record to determine what keywords to use. Also depending whether we want simple or complex query, we call diffent functions to form strings. ConstructCo Private ConstructNewsP This function constructs the mplexNP Func- ageURL keywords to be send to the KeyWord tion NewsPage site. (BF.Search) UnlikeConstructKeyWordStr ing which simply takes all the keywords from the title to form the query, this function will look at the results of BF 's pattern matching process and see if we are able to identify any specific company names or topics for constructing the queries. Since newspage works best when we have a company name, we'll use only the company name and Only if there is no company will we use topic. ConstructOv Private GoBackGroundF This function takes in as erallResult Func- inder input an array of strings (BF.Main) tion (stInStories) and a MeetingRecord which stores the information for the current meeting. Each element in the array stores the stories retrieved from each information source. The function simply constructs the appropriate output to send to Munin including a return message type to let Munin know that it is the BF

responding and also the original user_id and meeting title so Munin knows which meeting BF is talking about. ConnectAnd Public GoBackGroundF This function allows TransferTo Sub inder Background Finder to Munin connect to Munin and (BF.Main) eventually transport information to Munin. We will be using the UDP protocol instead of the TCP protocol so we have to set up the remote host and port correctly. We use a global string to store gResult Overall because although it is unnecessary with UDP, it is needed with TCP and if we ever switch back don't want to change code. DisconnectFr Public omMuninAn Sub Quit (BF.Main)

Detailed Description Paragraph Table (10):

```
Intention-Centric Interface Create an Intention ASP Page ("intention_create.asp")
<%@ LANGUAGE = "JScript" %> <% Response.Buffer = true; Response.Expires = 0; %>
<html> <head> <title>Create An Intention</title> </head> <body bgcolor="#FFE9D5"
style="font-family: Arial" text="#000000"> <% //Define some variables upl =
Server.CreateObject("SoftArtisans.FileUp") intention_name = upl.Form
("intention_name") intention_desc = upl.Form("intention_desc") //intention_name =
Request.Form("intention_name") //intention_desc = Request.Form
("intention_desc") //intention_icon = Request.Form("intention_icon") submitted =
upl.Form("submitted") items = new Enumerator(upl.Form) %> <% //Establish connection
to the database objConnection = Server.CreateObject ("ADODB.Connection")
ObjConnection.Open ("Maelstrom") %> <% //Check to see if the person hit the button
and do the appropriate thing if (submitted == "Add/Delete") { flag = "false" //loop
through all the inputs while(!items.atEnd( )) { i = items.item( ) //if items are
checked then delete them if(upl.Form(i) == "on") { objConnection.Execute("delete
from user_intention where intention_id =" + i); objConnection.Execute("delete from
intentions where intention_id =" + i); objConnection.Execute("delete from
tools_to_intention where intention_id =" + i) flag = "true" } items.moveNext
( ) } // if items were not deleted then insert whatever is in the text field in the
database if(flag == "false") { intention_name_short = intention_name.replace
(/ /gi," ") objConnection.Execute("INSERT INTO intentions
(intention_name,intention_desc,intention_icon) values(' " + intention_name + " ', '
" + intention_desc + " ',' " + intention_name_short + ".gif" + " ')" Response.write
("the intention short name is " + intention_name_short); upl.SaveAs
("E:development/asp_examples/"+ intention_name_short +".gif") } } // Query the
database to show the most recent items. rsCustomersList = objConnection.Execute
("SELECT * FROM intentions") %> <input type="Submit" name="return_to_mcp" value="Go
to Main Control Panel" onclick="location.href='default.asp' "> <form method="post"
action="intention_create asp" enctype="multipart/form-data" > <TABLE border=0>
<tr><td colspan="2"><font face="Arial" size="+1"><b>Enter in a new
intention</b></font></td></tr> <tr><td><font face="Arial">Name:</font></td>
<td><INPUT TYPE="text" name="intention_name"></td></tr> <tr><td><font
face="Arial">Description:</font></td><td><TEXTAREA
name="intention_desc"></TEXTAREA></td></tr> <tr><td><font face="Arial">Icon
Image:</font></td><td><INPUT TYPE="file" NAME="intention_icon" size=40></td></tr>
<tr><td colspan="2"><INPUT type="submit" name="submitted"
value="Add/Delete"></td></tr> </TABLE> <HR> <font face="Arial" size="+1"><b>Current
Intentions</b></font> <TABLE> <tr bgcolor=E69780 align="center"> <td> <FONT
color="white">Delete</FONT> </td> <TD> <FONT color="white">Itention</FONT> </TD>
<TD> <FONT color="white">Description</FONT> </TD> <TD> <FONT
color="white">Image</FONT> </TD> </tr> <% // Loop over the intentions in the list
counter = 0; while (!rsCustomersList.EOF) { %> <tr bgcolor="white" style="font-
size: smaller"> <td align=center> <INPUT type="checkbox" name="<%=rsCustomersList
("intention_id")%>"> </TD> <td> <%= rsCustomersList ("intention_name")%> </td> <td>
<%= rsCustomersList ("intention_desc")%> </td> <td> <img src=". ./images/<%=
rsCustomersList("intention_icon")%>"> </td> </tr> <% counter++
rsCustomersList.MoveNext ( )} %> </TABLE> <hr> Available Tools </form> </BODY>
</HTML> Retrieve Intentions List ASP Page ("intentions_list.asp") <!-- #include
file="include/check_authentication.inc" --> <HTML> <HEAD> <TITLE>mySite! Intentions
List</TITLE> <SCRIPT LANGUAGE="JavaScript"> function intentionsList ( )
{ this.internalArray = new Array( ); <% // establish connection to the database
```

```
objConnection = Server.CreateObject ("ADODB.Connection"); objConnection.Open
("Maelstrom"); // create query intentionsQuery = objConnection.Execute("SELECT *
FROM intentions ORDER BY intention_name asc"); %> // write out the options <%
numOptions = 0 while (!intentionsQuery.EOF) { intentionName = intentionsQuery
("intention_name"); intentionIcon = intentionsQuery("intention_icon"); %> .
this.internalArray[<%= numOptions%>] = new Array(2); this.internalArray[<%=
numOptions%>][0]= "<%= intentionName %>"; this.internalArray[<%= numOptions%>][1]=
"images/<%= intentionIcon %>"; <% numOptions++; intentionsQuery.moveNext( ); %>
<% } %> } numIntentions = <%= numOptions%>; intentionArray = new intentionsList
( ).internalArray; function selectIntention ( ) { for (i=0;i<numIntentions;i++)
{ if (IntentionsListSelect.options[i].selected) { intentionNameTextField.value =
intentionArray[i] [0]; //intentionPicture.src = intentionArray[i] [1]; break; } } }
</SCRIPT> </HEAD> <BODY BGCOLOR="<%=Session("main_background")%>" style="font-
family: Arial"> <CENTER> <!--- <FORM NAME="intention_list"> ---> <TABLE FRAME="BOX"
border=0 CELLPADDING="2" CELLSPACING="2"> <TR><TD COLSPAN="3" STYLE="font: 20pt
arial" ALIGN="CENTER"><B>Add a mySite! Intention</B></TD></TR> <TR><TD
COLSPAN="3"> </TD></TR> <TR> <TD width="100"><font size="-1">Please Select An
Intention You Would Like to Add to Your List</font></TD> <TD colspan=2> <SELECT
ID="IntentionsListSelect" NAME="IntentionsListSelect" SIZE="10" style="font: 9pt
Arial;" onClick="selectIntention( )"> <% intentionsQuery.moveFirst ( ); for
(j=0;j<numOptions;j++) { %> <OPTION VALUE="<%= intentionsQuery("intention_id") %>"
<% if (j == 0) { %> SELECTED <% } %>> <%=intentionsQuery("intention_name") %> <%
intentionsQuery.moveNext ( ) } intentionsQuery.moveFirst ( ); %> </SELECT> </TD>
</TR> <TR><TD COLSPAN="3">  </TD></TR> <TR> <TD width="100"><font size="-
1">Customize the Intention name</font></TD> <TD COLSPAN=2"><INPUT TYPE="text"
NAME="intentionNameTextField" ID="intentionNameTextField" SIZE="30" VALUE="<%=
intentionsQuery("intention_name") %>"></TD> </TR> <TR><TD COLSPAN="3">  </TD></TR>
<TR> <TD COLSPAN="3" ALIGN="CENTER"> <INPUT TYPE="button" NAME="intentionOKButton"
VALUE=" OK " SIZE="10" ID="intentionOKButton"
onClick="javaScript:top.opener.top.navframe.addAnIntention ( );">
 &nbs p; <INPUT TYPE="button" NAME="intentionCancelButton" VALUE="Cancel" SIZE="10"
ID="intentionCancelButton" onClick="self.close( ) ;"> </TD> </TR> </TABLE> <!---
</FORM> ---> </CENTER>

Detailed Description Paragraph Table (11):
<% objConnection.Close( ); %> </BODY> </HTML> Display User Intention List ASP Page
(excerpted from "navigation.asp") <DIV ID="intentionlist" style="position:
absolute; width:210; height:95; left: 365pt; top: -5; visibility: hidden; font-
family: Arial; font-color: #000000; font: 8pt Arial ; " > <DIV style="position:
absolute; top:7; left:7; height:78; width:210; z-index:2; background: <%=Session
("main_background")%>; border: solid 1pt #000000; padding: 3pt; overflow: auto;
alink: black; link: black;"> <body LINK="#000000" ALINK="#000000" vlink="black">
<% // create query intentionsQuery = objConnection.Execute("SELECT user_intention.*
FROM user_intention, user_intention_to_persona WHERE
user_intention_to_persona.user_persona_id = " + Session("currentUserPersona") + "
AND user_intention_to_persona.user_intention_id =
user_intention.user_intention_id"); numintentions = 0; Response.Write
("<SCRIPT>numintentions=" + intentionsQuery.RecordCount + "</SCRIPT><TABLE
cellpadding='0' width='100%' cellspacing='0'>"); while (!intentionsQuery.EOF) { %>
<TR><TD><a href="javascript:changeIntention('<%= intentionsQuery
("user_intention_id") %>', '<%=numintentions%>')" onmouseover="mouseOverTab ( )"
onmouseout="mouseOutOfTab( ) "><font color="Black" face="arial" size="-2"><%=
intentionsQuery("intention_custom_name") %></font></a></TD><TD><IMG align="right"
SRC="images/delete.gif" alt="Delete this intention" onClick="confirmDelete (<%
=intentionsQuery("user_intention_id") %>) "></TD></TR> <%numintentions++;
intentionsQuery.moveNext( ); %> <% { Response.Write
("<SCRIPT>numintentions="+numintentions +"</SCRIPT>"); %> <tr><td
colspan="2"><hr></td></tr> <TR><td colspan="2"><a href="javascript:changeIntention
('add . . . ',<%=numintentions%>);" onmouseover="mouseOverTab( )"
onmouseout="mouseOutOfTab( ) "><font color="Black" face="arial" size="-2">add . . .
```

```
</font></a></td></TR> </table> </body> </DIV> <DIV style="position: absolute;
top:0; left:-5; width: 230; height:105; z-index:1; "
onmouseout="intentionlist.style.visibility='hidden' "
onmouseout="intentionlist.style.visibility='hidden' "
onmouseover="intentionlist.style.visibility='hidden' "></DIV> </DIV> </DIV>
```


CLAIMS:

13. An apparatus for creating a user network interface which is accessible from a plurality of locations, comprising: a processor; a memory that stores information under the control of the processor; logic that identifies a user; logic that identifies in a database a plurality of stored profiles corresponding to the user logic that determines which of the identified plurality of stored profiles corresponds to a current location of the user; logic that identifies information of interest to the user based on the determined user profile; logic that prioritizes the information of interest to the user based on the determined user profile; logic that displays the information of interest on a web page formatted in accordance with the determined user profile; and logic that updates the determined user profile in the database based on interaction with the displayed information by the user in accordance with the current location of the user.

Previous Doc Next Doc Go to Doc#

L13: Entry 2 of 2 File: USPT Jan 15, 2002

DOCUMENT-IDENTIFIER: US 6339795 B1
TITLE: Automatic transfer of address/schedule/program data between disparate data hosts

Brief Summary Text (4):
The present invention relates generally to methods for exchanging data between disparate data hosts including application programs and data bases. More specifically, the present invention relates to a user transparent process for exchanging and routing data representing postal address information between disparate data hosts.

Brief Summary Text (6):
Application programs and databases, including relational databases, are examples of data hosts used for generating, manipulating, and storing data. A wide variety of data hosts are commercially available for managing many different types of data for a multitude of purposes. Application programs and databases typically include strict rules for defining composite data types that may be used therein. The data types may include records, arrays and other structures.

Brief Summary Text (7):
Generally, data formats may be categorized as either plain text data, or parsed and tagged data. Plain text data is of variable length and composition and is not easily parsed into fields, and therefore there are no portions of the plain text data which are separately identifiable. Plain text data is most commonly managed in word processing type application programs. In database files, data is generally managed in a parsed and tagged type of format either by a database manager or by a special purpose application program.

Brief Summary Text (8):
Database files generally include data records and header records. In general, database files may be managed either by a database manager or by a special-purpose application program. A database manager provides for a user to specify record structures upon creation of the database file. A record structure is generally described by field names, data formats, and byte offsets or specific delimiters in the record. Database manager programs maintain data dictionary records as headers in the database file, the records typically specifying parameters associated with each field including a name, a start byte offset, and a data format. Special-purpose application programs are used to generate and manipulate databases of one specified record structure, the specification of which is embedded in the code of the program rather than in header records of the file. Currently, there is no standard internal data format used by all application programs and data base managers. Application programs and data bases typically use complex proprietary data formats.

Brief Summary Text (9):
The disparity in internal data formats between different types of application programs and database managers causes problems for users who wish to exchange data between these disparate databases. A disparity in internal data formats from one

data host to another may also arise due to the use of different compilers and different hardware architectures, sometimes referred to as "platforms". application programs and data bases are written in a higher order language, and then compiled by other programs called compilers. The same or different compilers used on different computers may result in different internal data formats for the same data. Different compilers used on identical platforms may also result in different internal data formats. Another problem is that different compilers and platforms may use different byte ordering including Big-Endian and Little-Endian byte ordering.

Brief Summary Text (10):
It has become increasingly desirable for users to be able to conveniently exchange data between disparate application programs and databases running on disparate computer platforms including desk top computers, hand held computers, and web servers. Due to the disparities in the internal data formats of the various data hosts, transfer of data between disparate data hosts typically is not readily achievable via ordinary file transfer. The different internal data formats must be reconciled for disparate data hosts to communicate with each other. When information is to be exchanged between disparate data hosts, some form of data format conversion is required.

Brief Summary Text (11):
A variety of prior art techniques have been developed specifically for exchanging data between handheld computers and desk top computers. Handheld computers, such as personal digital assistants (PDA's), typically provide some combination of personal information management functions, database functions, word processing functions, and spreadsheet functions. Due to limitations in memory size and processing power, handheld computers are generally limited in functionality and differ in data content and usage from similar applications on desktop computers. Many users of handheld computers, such as personal digital assistants (PDA's), also own a desktop computer which may be used for application programs that manage data similar to the data stored in the handheld computer. A user typically stores the same data on the desktop computer and handheld computer. Therefore, it is very desirable for a user to be able to conveniently exchange data between desk top application programs and data bases, and memory resident data sets of a hand held computer.

Drawing Description Text (10):
FIG. 10B is a block diagram of pattern matching database modules used in accordance with the address data parsing sub-process of the present invention;

Detailed Description Text (2):
FIG. 1 shows a generalized block diagram of a networked system at 10 for implementing a process according to the present invention for automatic transparent exchange of data between disparate data hosts including, application programs and data base managers, having different internal data formats, wherein the disparate data hosts may be running on disparate computer platforms. The system 10 comprises: a plurality of user sites, or client sites, including a first user site 12, and a second user site 14 located remotely from the first user site 12 and coupled for communication with the first user site 12 via a network 16; a dedicated index web-site 18 having a dedicated index web-server 19 according to the present invention coupled for communication with at least one of the user sites via the network 16; and a web-site 20 having a web-server 21 coupled for communication with at least one of the user sites via the network.

Detailed Description Text (3):
The first user site 12 includes: a first client computer system 22; and hand-held computer devices 24 coupled with the computer system 22 via coupling means 26 (e.g., a cable or a bus). The second user site 14 includes: a second client computer system 23 providing a computer platform different from the platform provided by the first client computer system 22 of the first user site 12; and

hand-held computer devices 24 coupled with the computer system 23. The hand-held computer devices 24 may include, for example, a personal digital assistant (PDA) 28 (e.g., a Palm-Pilot.TM. device) and a pocket organizer 30. Each of the hand-held computer devices 24 provides personal information management functions, database functions, word processing functions, and spread sheet functions.

Detailed Description Text (8):
The computer readable memory 37 has computer executable code stored therein used for implementing: a client-side data exchange process module 60 for implementing client-side functions of the automatic transparent data exchange process according to the present invention; a plurality of client resident data hosts 62, including application programs and data bases, which communicate with the system bus 36 as indicated by a line 63, and which also communicate with the data exchange process module 60 as indicated by a line 64; a back-up storage driver module 66 which communicates with the data exchange process module 60 as indicated by a line 68; and a back-up storage unit 70 which communicates with the back-up storage driver module 66 as indicated by a line 72. The data exchange process module 60 communicates with the external device interface 44 and network interface 46 via the system bus 36 as indicated by a line 74. The client resident data hosts 62 are executed by the processing unit 34 (FIG. 2) of the client computer system.

Detailed Description Text (23):
Specialized communication methods are used for cases wherein one of the data hosts is a "form" provided by the web server 21 (FIG. 1) to the client computer system via the network 16 (FIG. 1). Typically, such a form is accessed via browser type application program executed by the client computer system. CGI provides a way for browsers on different platforms to interact with data bases on equally diverse platforms. Through CGI scripts, nearly every type of data access is possible. The general principles of data base access are the same for any web server that supports CGI. CGI scripts provide a means for passing data between web servers and other applications. Most data base gateways use CGI in some manner. Some web servers allow the use of dynamic data exchange (DDE) and object linking and embedding (OLE) in the windows environment to exchange data directly between a web server and various applications.

Detailed Description Text (24):
A difficulty with CGI is that it always requires programming. CGI is only an interface, or a front-end. The data is still contained in a data base on the back-end, or the part of the information system that is hidden from the user by the facade of the interface. In order to link the front-end and back-end, custom scripts must be written to link specific data bases to the generic interface. Retrieving data from a back-end data base can be done in one of two ways. The simplest is to read the data base files directly in their native format. If this is not possible, the CGI program must communicate with the data base server. If the data base files cannot be read directly, it is necessary to communicate with the data base server, which reads the files and sends the results back to the client (CGI program). This is only possible for data bases that implement a standards-based server such as in a structured query language (SQL) server or open data base connectivity (ODBC) server. In this manner, any SQL or ODBC client can communicate with the data base server. Nearly all data bases are either SQL or ODBC compatible including Informix, SyBase, Oracle, Borland, Paradox and InterBase, Microsoft Access, and Lotus Approach.

Detailed Description Text (33):
In the case in which the data host is a database file, the host detection and invocation API 162 is used to determine if the database file exists and, if it does, provides for loading it. In some cases like a driver for a handheld device, it might not be possible to use the invocation function of the API 162 in which case the API 162 will return without doing anything, except setting an appropriate error code for subsequent handling and reporting to the operator.

Detailed Description Text (35):
The host specific setting dialog and menu invocation API 164 provides for the user to specify settings for each driver interface module. For example, in the case wherein the data host is an address book application program, the user may instruct the driver interface module to use a home phone number as a main phone and disregard any other phone number. In the case wherein the data host is a database, the user may specify which database file the driver is to open for executing a data exchange. These settings may be specified either via a settings dialog, or via context menus which appear on the driver icon on the display device 42 (FIG. 2). Using the API 164, the user may also obtain icons representing each driver interface module. The icons are then shown on the toolbar on the display device for quick access.

Detailed Description Text (39):
Because there is such a vast assortment of commercially available application programs and data bases, it is prohibitive in both time and cost to accommodate all potential combinations of driver interface modules. Therefore, the data exchange process provides: a method for determining the file format characteristics of a data host which is not specifically supported; and means for determining and providing a generic driver module, or interface, for interfacing with the data host which is not specifically supported.

Detailed Description Text (61):
As further described below, the parsing module includes computer readable instructions which, when executed, perform the steps of: reading the plain text data which has been extracted from the source host; performing preprocessing functions to presort the plain text data into simplified text lines; determining patterns, or text strings, of the data block which match entries stored in a plurality of pattern matching data bases, or libraries; generating a matching probability table including a plurality of probability weight factors indicating, for each of a plurality of identified text strings of the plain text data, the probability that the corresponding text string represents a particular type of information; and processing the plain text data in accordance with a plurality of contextual analysis sub-processes which modify the probability weight factors stored in the matching probability table in order to increase accuracy in determining the probabilities that the text strings represent the particular corresponding types of information.

Detailed Description Text (64):
As mentioned, one stage of the address parsing and tagging process includes "pattern matching" in which patterns, or text strings, of the plain text data are compared with entries stored in a plurality of pattern matching data bases, or libraries. Each of the pattern matching databases includes a library of entries including characters and elements to be searched for in the text strings or the plain text data. Entries may include particular words, and particular patterns. In the preferred embodiment, there are eleven pattern matching databases, all of which are loaded from a single database file, "PARSER.DB", which is located in a parser directory. In the preferred embodiment, the parser is encrypted in binary form to prevent reverse engineering. In an alternative embodiment, this file is an ASCII file which makes it very easy to edit the patterns.

Detailed Description Text (65):
FIG. 10B shows a block diagram depicting at 370 a plurality of pattern matching databases stored in the data parsing and tagging module 106 (FIG. 3) of the data exchange program module 80. The databases include: a negative name matching database 372 which includes a list of words which have a very low probability of occurring in names (e.g., sales, marketing, world, help, orange, etc.) and which are used to determine negative name matches which substantially decrease the probability that a text string matching an entry in this data base is a name; a

positive name pattern matching database 374 including name entries for which a match with a text string suggests, with some predetermined probability weight factor, that the matching text sting is a name; a country name pattern matching data base 375 including country name entries for which a match with a text string suggests with some predetermined probability weight factor that the matching text string is a country name, the country entries including all country names and abbreviations thereof; a company name pattern matching database 376 including company entries for which a match suggests that the matching text sting is a company name the company entries including standard company endings (e.g., "Inc.", "company", Ltd., etc.) and also the names of Fortune 500 companies; a title name pattern matching database 378 including title entries for which a match suggests that the matching text sting is a title, the title entries including common titles (e.g., manager, CEO, administrator, etc.); a state name pattern matching database 380 including entries for which a match suggests that the matching text sting is a state name, the state entries including all full state names and state abbreviations (e.g., California and CA); a city name pattern matching database 382 including entries for which a match suggests that the matching text sting is a city name; an address name pattern matching database 384 including entries for which a match suggests that the matching text sting is a "street address"; a zip code pattern matching database 386 including entries for which a match suggests that the matching text sting is a zip code; an e-mail pattern matching database 384 including entries for which a match suggests that the matching text sting is an e-mail address; a phone number pattern matching database 390 including entries for which a match suggests that the matching text sting is a phone number; a facsimile number pattern matching database 392 including entries for which a match suggests that the matching text sting is fax number; a web address pattern matching database 394 including entries for which a match suggests that the matching text sting is a web address; an amount pattern matching database 396 including entries for which a match suggests that the matching text sting is an amount; and a date pattern matching database 398 including entries for which a positive match suggests that the matching text sting is a date. In one embodiment, a different set of pattern matching databases is used for each of a plurality of countries or geographical regions. The default country is USA. For example, the set of data bases for the United States includes data bases having English language entries.

Detailed Description Text (70):
In step 422, the parsing module: loads the country name pattern matching database 375 (FIG. 10B); reads the plain text data; and compares text strings of the plain text data to entries in the country name pattern matching database to determine if a country pattern match exists for the plain text data. If no country match is determined, the parsing module assumes that the plain text data includes an address of a particular country (e.g., a United States address). In step 424, based on the country determined in step 422, the parsing module loads an appropriate set of pattern matching data bases 372-398 (FIG. 10B) into the working memory unit of the client computer system.

Detailed Description Text (71):
In step 426, the parsing module determines positive and negative matches for the plain text data by determining all text strings of the plain text data which match an entry of any one of the pattern matching databases 372, 374, 376, 378, 380, 382, 384, 386 (FIG. 10B). In step 428, the parsing module generates a matching probability table 350 (FIG. 10A) in the memory unit 37 (FIG. 2) of the client computer system wherein the probability weights stored in the probability weight columns 358 (FIG. 10A) in the table are initialized based on the positive and negative matches determined for the plain text data in step 426. From step 428, the process proceeds to "P1" (to FIG. 11B).

Detailed Description Text (77):
In step 452, the parsing module performs pattern matching using the e-mail address matching data base 388 (FIG. 10B) to determine if the plain text data includes an

e-mail match. In step 454, the parsing module performs matching using the amount pattern matching data base 396 (FIG. 10B) to determine whether the plain text data includes a text string having a high probability of representing an amount. If a text string constituting an amount match is found, an amount tag is associated with the corresponding text string.

Detailed Description Text (78):
In step 456, the parsing module performs pattern matching using the date pattern matching data base 398 (FIG. 10B) to determine whether the plain text data includes a date match which is a text string having a high probability of representing a date (e.g., Jan. 1, 1999). If a date match is found, a date tag is associated with the corresponding text string.

Detailed Description Text (79):
In step 458, the parsing module performs phone number pattern matching using the phone number pattern matching data base 390 (FIG. 10B) to determine whether the plain text data includes a phone number match which is a text string having a high probability of being a phone number, such as a cell phone number, a pager number, etc. If a phone number match is found, a phone number tag is used to designate the corresponding text string as a phone number.

Detailed Description Text (80):
At 460, the parsing module determines whether a phone number match has been determined in accordance with step 458, and if so, the process proceeds to step 462 in which the parsing module performs facsimile number pattern matching using the data base 392 (FIG. 10B) to determine whether the plain text data includes a facsimile number match which is a text string having a high probability of being a facsimile number. From step 462, the process proceeds to "P2" (to FIG. 11C). If a phone number match has not been determined, the process proceeds from 460 directly to "P2" (to FIG. 11C).

Detailed Description Text (86):
If it is determined at 510 that more than one of the zip code type patterns searched for in steps 502 and 504 has been identified, the process proceeds to step 514 in which it is determined whether any of the identified zip code type patterns is located to the right of or below a text string determined in step 426 (FIG. 11A) to be state pattern match, that is a text string matching an entry in the state name pattern matching database 380 (FIG. 10B). Also in step 514, the parsing module accords positive zip code matching weight factors for the identified zip code type patterns. These weight factors are added to cumulative zip code probability weights stored in the corresponding one of the columns 358 of the matching probability table 350 (FIG. 10). A zip code type pattern which is located to the right of a state pattern match is accorded a higher zip code matching weight factor than a zip code type pattern located below a state pattern match.

Detailed Description Text (91):
FIG. 13A shows a flow diagram at 550 depicting the name matching contextual analysis sub-process invoked by the parsing module in step 442 (FIG. 11B). The depicted sub-process begins at 552 in which the parsing module determines whether a positive name pattern match, that is a text string matching a name entry in the positive name matching database 374 (FIG. 10B), has been identified in step 426 (FIG. 11A). If the plain text data includes a positive name pattern match, the process proceeds from 552 to step 554 in which the parsing module removes negative name pattern matches, that is text strings which match entries in the negative name matching database 372 (FIG. 10B), from the text lines including positive name pattern matches.

Detailed Description Text (96):
In step 584, the parsing module accords a positive name matching probability weight to those text lines which do not include a pattern match, and which include two or

three words, and which are located above a text line including a title pattern match or a company pattern match as determined in step 426 (FIG. 11A). In step 586, the parsing module adds a positive name matching probability weight for those text lines which match an entry in the positive name database. Note that this database is updated if the user chooses to correct a wrongly recognized name, so that the same mistake is not made again.

Detailed Description Text (98):
In step 590, the parsing module reduces the name matching probability weight associated with those text lines which are located after an address pattern match. In step 592, the parsing module zeros out the name matching probability weights associated with all text strings including two or more numeric digits. In step 594, the parsing module zeros out the name matching probability weights associated with all text strings for which a non-name database pattern match occurs in the text string. In step 596, the parsing module determines the text string having the largest name matching probability weight.

Detailed Description Text (100):
FIG. 14 shows a flow diagram at 620 illustrating the state name contextual analysis sub-process invoked by the parsing module in step 436 (FIG. 11B) of the address data parsing process. In step 622, the parsing module adds a negative state name matching probability weight for all state pattern matches, determined in step 426 (FIG. 11A), having a two letter abbreviation (e.g., CA) wherein the first letter is not the same case as the second letter (e.g., cA or Ca). In step 624, the parsing module adds a large positive state name matching probability weight for those state pattern matches located on the same text line as a zip code type pattern match, or on a text line preceding a zip code type pattern match. In step 626, the parsing module determines if a zip code pattern match has been determined in step 426 (FIG. 11A), and if so, the parsing module consults a reverse zip code to state data base. This reverse data base is used to determine if a state pattern match indicates the same state which is indicated by an entry in the reverse data base corresponding with the zip code pattern match, and if so, the parsing module adds a large positive weight for the state pattern match. If not, the parsing module adds a negative weight for the state pattern match.

Detailed Description Text (101):
In step 628, the parsing module determines whether a phone number pattern has been determined in step 426 (FIG. 11A) and whether an area code is included in the phone number pattern. If a phone number pattern having an area code is found, the parsing module consults a reverse area code to state data base, and determines if a state pattern match indicates the same state indicated by an entry in the reverse data base corresponding with the area code. If so, the parsing module adds a large positive state matching probability weight. If not, the parsing module adds a negative weight for the state pattern match.

Previous Doc Next Doc Go to Doc#